JAVA

SEJARAH SINGKAT PERKEMBANGAN JAVA

- **Proyek Java dimulai pada tahun 1991**
  - sejumlah insinyur perusahaan Sun yang dimotori oleh ***James Gosling*** mempunyai keinginan untuk mendesain sebuah *bahasa komputer kecil* yang dapat dipergunakan untuk peralatan konsumen seperti kotak tombol saluran TV.
  - Proyek ini kemudian diberi nama sandi ***Green***.
- **Keharusan untuk membuat bahasa yang kecil dan kode yang ketat**
  - mendorong mereka untuk menghidupkan kembali model yang pernah dicoba oleh bahasa UCSD Pascal, yaitu mendesain sebuah *bahasa yang portable* yang menghasilkan **kode intermediate**.
  - Kode intermediate ini kemudian dapat digunakan pada banyak komputer yang *interpreternya telah disesuaikan*.
- **Karena orang-orang Sun memiliki latar belakang sebagai pemakai unix**
  - lebih menggunakan C++ sebagai basis bahasa pemrograman mereka, maka mereka secara khusus mengembangkan *bahasa yang berorientasi objek* bukan berorientasi prosedur.
  - Gosling : ”*Secara keseluruhan, bahasa hanyalah sarana, bukan merupakan tujuan akhir”*.

    Gosling memutuskan menyebut bahasanya dengan nama “Oak” (diambil dari nama pohon yang tumbuh tepat diluar jendela kantornya di Sun), tetapi kemudian nama Oak diubah menjadi java, karena nama Oak merupakan nama bahasa komputer yang sudah ada sebelumnya.

Pada tahun 1994 sebagian besar orang menggunakan **mosaic**, browser web yang tidak diperdagangkan yang berasal dari pusat *Supercomputing* Universitas Illinois pada tahun 1993. (Mosaic sebagian ditulis oleh Marc Andreessen dengan bayaran $6.85 per jam, sebagai mahasiswa yang melakukan studi praktek. Di kemudian hari ia meraih ketenaran sebagai salah seorang pendiri dan pemimpin teknologi di netscape)

Browser yang sesungguhnya dibangun oleh **Patrick Naughton** dan **Jonathan Payne** dan berkembang ke dalam browser HotJava yang kita miliki saat ini. Browser HotJava ditulis dalam Java untuk menunjukkan kemampuan Java.

Tetapi para pembuat juga memiliki ide tentang suatu kekuatan yang saat ini disebut dengan **Applet**, sehingga mereka membuat browser yang mampu penerjemahkan **Kode Byte** tingkat menengah. “Teknologi yang Terbukti” ini diperlihatkan pada SunWorld ’95 pada tanggal 23 mei 1995, yang mengilhami keranjingan terhadap Java terus berlanjut.

KRITERIA “KERTAS PUTIH” JAVA

Penulis Java telah menulis pengaruh “Kertas Putih” yang menjelaskan tujuan rancangan dan keunggulannya. Kertas mereka disusun lewat 11 kriteria berikut :

1. Sederhana (Simple)
2. Berorientasi Objek (Object Oriented)
3. Terdistribusi (Distributed)
4. Kuat (Robust)
5. Aman (Secure)
6. Netral Arsitektur (Architecture Neutral)
7. Portabel (Portable)
8. Interpreter
9. Kinerja Yang Tinggi (High Performance)
10. Multithreaded
11. Dinamis

SEDERHANA (SIMPLE)

Syntax untuk Java *seperti* syntax pada C++ tetapi syntax Java **tidak memerlukan :**

- header file
- pointer arithmatic (atau bahkan pointer syntax)
- struktur union
- operator overloading
- class virtual base
- dan yang lainnya.

Jika anda mengenal C++ dengan baik, maka anda dapat berpindah ke syntax Java dengan mudah tetapi jika tidak, anda pasti tidak berpendapat bahwa Java sederhana.

BERORIENTASI OBJEK (OBJECT ORIENTED)

Rancangan berorientasi objek merupakan suatu teknik yang *memusatkan rancangan pada **data (objek)** dan **interface***. Fasilitas pemrograman berorientasi objek pada Java pada dasarnya adalah sama dengan C++.

Feature pemrograman berorientasi objek pada Java benar-benar sebanding dengan C++.

Perbedaan utama antara Java dengan C++ terletak pada penurunanberganda (multiple inheritance), untuk ini Java memiliki cara penyelesaian yang lebih baik.

TERDISTRIBUSI (DISTRIBUTED)

Java memiliki *library rutin yang luas* untuk dirangkai pada protokol TCP/IP sepetrti HTTP dan FTP dengan mudah.

Aplikasi Java dapat *membuka dan mengakses objek* untuk segala macam NET lewat URL sama mudahnya seperti yang biasa dilakukan seorang programmer ketika *mengakses file sistem secara lokal*.

KUAT (ROBUST)

Java dimaksudkan untuk membuat suatu program yang benar-benar dapat dipercaya dalam berbagai hal.

Java banyak menekankan pada :

- *pengecekan awal* untuk kemungkinan terjadinya masalah
- *pengecekan pada saat runtime*
- *mengurangi kemungkinan timbulnya kesalahan* (error)

Perbedaan utama antara Java dan C++ adalah Java memiliki sebuah *model pointer* yang *mengurangi kemungkinan penimpaan* (overwriting) pada memory dan *kerusakan data* (data corrupt).

AMAN (SECURE)

Java dimaksudkan untuk digunakan pada jaringan terdistribusi. Sebelum sampai pada bagian tersebut, penekanan terutama ditujukan pada *masalah keamanan*.

Java memungkinkan penyusunan program yang

- bebas virus
- sistem yang bebas dari kerusakan.

NETRAL ARSITEKTUR (ARCHITECTURE NEUTRAL)

Kompiler membangkitkan sebuah format file dengan objek arsitektur syaraf.

Program yang di kompile dapat dijalankan pada banyak prosesor, disini diberikan *sistem run time* dari Java.

Kompiler Java melakukannya dengan *membangkitkan instruksi-instruksi kode byte* yang tidak dapat dilakukan oleh arsitektur komputer tertentu.

Java dirancang untuk *mempermudah penterjemahan pada banyak komputer* dengan mudah dan diterjemahkan pada komputer asal *pada saat run-time*.

PORTABEL (PORTABLE)

Tidak seperti pada C dan C++, di Java terdapat ketergantungan pada saat implementasi *(implement dependent)*. Ukuran dari tipe data primitif ditentukan, sebagaimana kelakuan aritmatik padanya.

Librari atau pustaka merupakan bagian dari sistem yang mendefinisikan interface yang portabel.

INTERPRETER

Interpreter Java dapat *meng-eksekusi kode byte Java secara langsung* pada komputer-komputer yang memiliki interpreter.

Dan karena proses linking dalam Java merupakan proses yang kenaikannya tahap demi tahap dan berbobot ringan, maka *proses pengembangan dapat menjadi lebih cepat* dan masih dalam penelitian.

KINERJA YANG TINGGI (HIGH PERFORMANCE)

Meskipun *kinerja kode byte yang di interpretasi biasanya lebih dari memadai*, tetapi masih terdapat situasi yang memerlukan kinerja yang lebih tinggi.

Kode byte dapat diterjemahkan (pada saat run-time) ke dalam kode mesin untuk CPU tertentu dimana aplikasi sedang berjalan.

MULTITHREADED

Multithreading adalah kemampuan sebuah program untuk melakukan *lebih dari satu pekerjaan sekaligus.*
Keuntungan dari multithreading adalah *sifat respons yang interaktif dan real-time*.

DINAMIS

Dalam sejumlah hal, Java merupakan bahasa pemrograman yang *lebih dinamis* dibandingkan dengan C atau C++.
Java dirancang untuk *beradaptasi dengan lingkungan yang terus berkembang*.
Librari dapat dengan mudah *menambah metode dan variabel contoh yang baru* tanpa banyak mempengaruhi klien.
Informasi run-time dalam Java adalah langsung *(straigtforward)*.

BAGAIMANA JAVA LEBIH BAIK DARIPADA C++ ?

Prinsip dasar pembuatan Java adalah karena C++ ternyata tidak memenuhi janji sebagai pemrograman berorientasi objek. Jadi apa yang salah dari C++ sehingga Java harus dibuat ? Jawabannya sederhana, yaitu ***Kompatibilitas ke belakang*** *(backward compability)*.

Kompabilitas kebelakang biasanya dikenal sebagai kemampuan yang menjamin keberhasilan dengan membuat programmer belajar dengan cepat. Java menggunakan hampir semua konvensi yang identik untuk :

- deklarasi variabel
- melewatkan parameter
- operator
- pengaturan aliran.

Sehingga dengan kata lain Java menambahkan bagian-bagian yang baik dari C++ dan menghapus bagian-bagian yang jelek dari C.

Java jauh lebih baik dari C++ karena hal-hal yang tidak dimilikinya, seperti beberapa contoh berikut:

a) Variabel Global
b) Goto
c) Pointer
d) Alokasi Memori
e) Tipe Data Yang Rapuh
f) Pemilihan Tipe (Type Casting) yang Tidak Aman
g) Daftar Argumen Yang Tidak Aman
h) File Header yang Terpisah
i) Struktur yang Tidak Aman
j) Peng-hacker-an Preprocessor
k) QED

VARIABEL GLOBAL

Para programmer menulis program dalam bahasa assembly, dan semua program yang disimpan dalan punch card, penghubung alat pemrograman adalah variabel global, masalahnya, dengan menggunakan variabel global suatu fungsi dapat *memberikan efek samping yang buruk dengan mengubah keadaan global*.

Variabel global pada C++ adalah tanda sebuah *program yang tidak dirancang cukup baik* untuk enkapsulasi data dengan cara yang masuk akal.

Pada Java, *ruang penamaan global hanya hirarki class*.

Tidak mungkin menciptakan variabel global diluar semua class.

Setidaknya penentuan keadaan global dibuat lebih jelas dengan enkapsulasi dalam class.

Contoh, *system.out.println()* sering digunakan dalam program Java.

Ini adalah cara mengakses output standar global untuk interpreter Java.

GOTO

Beberapa kemampuan yang digunakan sebagai cara cepat untuk menyelesaikan program tanpa membuat struktur yang jelas adalah pernyataan *goto*.

Dalam C++ dikenal sebagai if-then-goto.

Sebelum C++ memasukkan penanganan eksepsi, goto sering digunakan untuk membuat perulangan di dalam keadaan eksepsi.

Java tidak memiliki pernyataan **goto.**

Java menyediakan kata **goto** hanya untuk *menjaga agar programmer tidak bingung menggunakannya*.

Java memiliki bagian **break** yang diberi label dan pernyataan **continue** yang merupakan bagian dimana goto boleh dipergunakan.

Penanganan eksepsi yang ampuh dan terdefinisi dengan baik pada Java menghilangkan kebutuhan perintah goto.

POINTER

Pointer atau address pada memori adalah kemampuan C++ yang paling ampuh juga paling *berbahaya.*

Biasanya kesalahan terjadi karena *"kurang satu tempat"* atau *rusaknya data* yang disimpan karena lokasi memori terakhir hancur - susah untuk diperiksa dan ditelusuri.

Meskipun penanganan objek Java menggunakan pointer, bahasa Java *tidak memiliki kemampuan memanipulasi pointer secara langsung.*

Kita tidak dapat mengubah integer menjadi pointer, menunjuk ulang sembarang address memori.

Array merupakan objek yang didefinisikan, tidak berupa address dimemori.

Di Java kita tidak dapat menulis sebelum akhir lokasi yang disediakan untuk array.

ALOKASI MEMORI

Kemampuan C++ yang sama berbahayanya dengan pengolahan matematis pointer adalah *manajemen memori*.

Manajemen memori di C dan C++ diwujudkan dengan keunggulan dan kelemahan fungsi library *malloc()* dan *free()*.

Fungsi **malloc()**, mengalokasikan jumlah tertentu memori (dalam byte), dan mengeluarkan address blok tersebut.

Fungsi **free()**, mengirimkan blok yang telah dialokasikan kepada sistem untuk penggunaan umum. Secara umum dapat menyebabkan *kebocoran memori* yang mengakibatkan program berjalan semakin lama semakin lambat.

Java tidak memiliki fungsi malloc dan free, karena setiap struktur data yang rumit adalah objek, maka mereka dialokasikan dengan **operator *new***, yang mengalokasikan ruang untuk objek pada 'heap' memori.

Memori yang disediakan disebut **'heap'** karena kita tidak perlu lagi memikirkannya sebagai penambahan address yang berstruktur linier.

Jadi hanya berupa *kumpulan instans (instance) objek.*

Yang didapat dari fungsi *new* bukanlah address memori, melainkan hanya *'pegangan' untuk objek* dalam heap.

TIPE DATA YANG RAPUH

C++ mewarisi semua tipe data umum pada C.

Tipe-tipe ini mewakili bilangan bulat dan pecahan dengan berbagai rentang nilai dan ketelitian. Rentang nilai dan ketelitian tipe ini bervariasi bergantung pada implementasi kompilernya.

Java memecahkan masalah ini dengan *mengambil ukuran yang sesuai untuk semua tipe numerik dasar dan menyatukannya.*

Arsitektur tertentu akan mengalami kesulitan atau bekerja tidak optimal untuk meng-implementasikan tipe data yang bergantung hardware secara ketat pada interpreter Java yang diberikan, tetapi inilah satu-satunya cara untuk menjamin hasil yang dapat dibuat ulang pada platform hardware yang berbeda.

PEMILIHAN TIPE (TYPE CASTING) YANG TIDAK AMAN

Type Casting adalah mekanisme yang ampuh dalam C/C++ yang memungkinkan kita untuk *mengubah tipe suatu pointer secara sembarang*.

Mungkin kita sering melihat bentuk seperti ini :

```
memset((void *)p, 0, sizeof (struct p))
```

Penggunaan ini, walaupun tidak baik, tetapi cukup aman. Tentu saja dengan menganggap blok memori yang ditunjuk oleh p sekurang-kurangnya sepanjang sizeof (struct p).
Ini harus digunakan dengan sangat hati-hati karena tidak ada syarat untuk memeriksa apakah kita telah memilih tipe dengan benar.
Penanganan objek Java mencakup informasi lengkap tentang class yang menjadi instans suatu objek, sehingga dapat dilakukan *pemeriksaan kompatibilitas tipe selama program berjalan*, dan *menghasilkan eksepsi jika terjadi kegagalan*.

DAFTAR ARGUMEN YANG TIDAK AMAN

C++ banyak disukai karena kemampuannya melewatkan pointer dengan tipe sembarang dalam daftar argumen panjang-variabel yang dikenal sebagai *varargs*.
Varargs adalah tambahan sederhana pada premis yang menyatakan bahwa *sembarang address dapat dipetakan pada sembarang tipe*, tugas pemeriksaan tipe diserahkan kepada programmer.
Sangat menyenagkan jika Java memiliki cara yang aman terhadap tipe untuk mendeklarasikan dan melewatkan daftar argumen panjang-variabel, tetapi sampai versi 1.0 belum ada ketentuan seperti itu.

FILE HEADER YANG TERPISAH

Salah satu kemampuan yang patut dipertimbangkan adalah file header, dimana kita dpata mendeklarasikan prototipe class kita dan mendistribusikannya dengan kode biner implementasi class yang telah di-compile. Kemampuan ini membuat lingkungan compiler C++ hampir tidak dapat digunakan. C++ memiliki format file yang bergantung mesin untuk kode yang telah di-compile, sehingga informasi header dapat dibuat coresiden. Karena antarmuka programmer ke class yang di-compile dilakukan melalui file header-nya, maka kode yang telah di-compile sangat bergantung pada apa yang ada pada file header tersebut.

Misalkan programmer yang senang berpetualang ingin meningkatkan akses pada beberapa anggota data private pada class yang telah di-compile. Yang harus dilakukan oleh orang tersebut adalah mengganti pengubah akses yang asalnya private menjadi public pada file header dan meng-compile suatu sub class dari class yang telah di-compile. Pada Java ini tidak mungkin terjadi, karena di Java tidak ada file header. Tipe dan visibilitas anggota class dicompile ke dalam file class Java. Interpreter Java menjalankan pengaturan akses saat program berjalan, jadi sama sekali tidak ada cara untuk mengakses variabel private dari luar suatu class.

STRUKTUR YANG TIDAK AMAN

C berusaha menyediakan *enkapsulasi* data melalui deklarasi struktur yang disebut *struct*, dan *polimorfisme* dengan mekanisme *union*. Dua gagasan ini menghasilkan batas tipis antara penyesuaian bergantung mesin yang kritis dan berbahaya dengan batasan ukuran.
Java tidak memiliki konsep *struct* dan *union* , sebaliknya Java menyatukan konsep ini dengan *class*.

PENG-HACKER-AN PREPROCESSOR

Untuk mewujudkan keinginan memiliki model yang jelas untuk ditulis oleh programmer, compiler C dan C++ menggunakan tool yang sama dengan yang digunakan pada masa-masa MACRO assembler.
Ini menghasilkan *preprocessor* C yang tugasnya mencari perintah khusus yang diawali tanda pagar (#).
Preprocessor C sering digunakan untuk *membangun program yang sangat sulit dibaca*.
Java mengatur agar kita *dapat bekerja tanpa preprocessor*, hanya bergantung pada kata kunci *final* untuk mendeklarasikan konstanta yang sebelumnya dihasilkan dengan *#define*.

QED

Berasal dari bahasa latin *Quod Erat Demonstrandum*, yang berarti “Terbuktikan..!!!”.

TATA BAHASA

Program Java adalah kumpulan :

1. Spasi
2. Komentar
3. kata kunci
4. identifier
5. literal
6. operator
7. pemisah.

SPASI

Java adalah bahasa bebas bentuk. Tidak perlu mengatur tata letaknya agar dapat bekerja.
Asalkan ada *sekurang-kurangnya satu* ***spasi, tab****, atau* ***baris baru*** *diantara setiap token sebelum disisipi operator atau pemisah lain.*

KOMENTAR

1. Komentar baris tunggal
   Diawali dengan tanda // dan diletakkan diakhir baris yang diberi komentar.
2. Komentar baris banyak
   Diawali dengan tanda /* dan ditutup dengan */ semua diantara kedua tanda tersebut dianggap komentar dan akan diabaikan oleh compiler.
   contoh penulisan :

```
/*
*       komentar……
*/
```

3. Komentar terdokumentasi
   Menggunakan piranti ***Javadoc***, yang mennggunakan komponen compiler Java untuk secara otomatis menghasilkan ***dokumentasi antar muka public*** suatu class.
   Aturan pembuatan komentar yang dapat diolah oleh Javadoc adalah :
   sebelum deklarasi class, method, dan variabel public harus digunakan komentar bertanda /** untuk menyatakan komentar dokumentasi, diakhiri dengan tanda */.
   Javadoc akan mengenali sejumlah variabel khusus yang *didahului dengan tanda* @ didalam bagian komentar.
   contoh penulisan :

```
/**
*       komentar….
*/
```

KATA KUNCI SIMPANAN (KEYWORDS)

Kata kunci simpanan adalah *identifier khusus* yang disimpan oleh bahasa Java untuk *mengendalikan bagaimana program didefinisikan*.

Kata kunci ini digunakan untuk mengenali :

- tipe-tipe
- pengubah
- mekanisme pengaturan aliran program.

Kata kunci ini hanya dapat digunakan ***untuk fungsi tertentu*** dan tidak dapat digunakan sebagai identifier nama suatu variabel, class dan method.

Sampai dengan Versi 1.0 terdapat 59 kata kunci seperti terlihat dalam tabel :

abstract	boolean	break	byte	byvalue	case
cast	catch	char	class	const	continue
default	do	double	else	extends	false
final	finally	float	for	future	generic
goto	if	implements	import	inner	instanceof
int	interface	long	native	new	null
operator	outer	package	private	protected	public
rest	return	short	static	super	switch
synchronized	this	throw	throws	transient	true
try	var	void	volatile	while	

IDENTIFIER

Digunakan untuk nama : Class, Method, dan Variabel.

Suatu variabel dapat berupa :

- urutan tertentu huruf (besar atau kecil)
- angka
- garis bawah
- tanda dolar
- Tidak boleh diawali oleh angka
- bersifat *case sensitive*.

Kelompok Java mengikuti aturan penamaan identifier untuk semua **method public** dan **variabel instans** dengan *huruf awal kecil* dan menandai bagian kata *selanjutnya dengan huruf besar*, misalnya **nextItem**, **currentValue**, **getTimeOfDay**.

Untuk variabel **private** dan **lokal** identifier akan berupa *huruf kecil semua* dikombinasikan dengan *garis bawah*, misalnya **next_val**, **temp_val**.

Untuk variabel **final** yang mewakili suatu konstanta, digunakan *huruf besar semua*, misalnya **TOK_BRACE**, **DAY_FRIDAY**.

LITERAL

Besaran konstanta pada Java dihasilkan dengan menggunakan literal yang mewakilinya. *Setiap literal merepresentasikan nilai* suatu tipe, dimana *tipe itu sendiri menjelaskan bagaimana sifat nilai* tersebut dan bagaimana penyimpanannya.

SEPARATOR (PEMISAH)

Simbol	Nama	Fungsi
()	Kurung	Digunakan untuk *menghimpun parameter* dalam definisi dan *pemanggilan method*, juga digunakan untuk menyatakan *tingkatan pernyataan*, *menghimpun pernyataan* untuk pengaturan alur program dan menyatakan *tipe cast*.
{}	kurung kurawal	Digunakan untuk *menghimpun nilai* yang otomatis dimasukkan *kedalam array*, juga digunakan untuk mendefinisikan *blok program*, untuk *cakupan class, method, dan lokal.*
[]	kurung siku	Digunakan untuk menyatakan *tipe array*, juga digunakan untuk *membedakan nilai array*.
;	titik-koma	pemisah pernyataan.
,	koma	*Pemisah urutan identifier* dalam deklarasi variabel, juga digunakan untuk mengaitkan pernyataan didalam pernyataan *for*.
.	titik	Digunakan untuk *memisahkan nama paket* dari sub-paket dan class, juga digunakan untuk *memisahkan variabel atau method* dari variabel referensi.

VARIABEL

Variabel adalah ***satuan dasar penyimpanan*** dalam program Java.

Suatu variabel didefinisikan dengan kombinasi

- Identifier
- Tipe
- cakupan.

Bergantung pada tempat kita mendeklarasikannya, variabel dapat bersifat lokal atau sementara, misalnya didalam perulangan *for*, atau dapat juga berupa variabel instans yang dapat diakses oleh semua method dalam class. Cakupan **lokal** dinyatakan dalam **kurung kurawal.**

OPERATOR

ASSIGNMENT OPERATOR (=)

Shorthand Assignment operator

Operator	Usage	Meaning
+=	X += Y	X = X + Y
-=	X -= Y	X = X – Y
*=	X *= Y	X = X * Y
/=	X /= Y	X = X / Y
%=	X %= Y	X = X % Y

ARITHMETIC OPERATOR

Operator	Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulo

BITWISE OPERATOR

Operator	Operation
&	AND
|	OR
^	XOR
>>	Shift Kanan
<<	Shift Kiri
>>>	Shift Kanan isi dengan nol

UNARY OPERATOR

Operator	Operation
~	Unary NOT
-	Minus
++	Increment
--	Decrement

RELATIONAL OPERATOR

Operator	Operation
==	Equal To
!=	Not Equal To
>	Greater Than
<	Less Than
>=	Greater or Equal To
<=	Less or Equal To

LOGICAL OPERATOR

Operator	Operation
!	Short-circuit NOT
&&	Short-circuit AND
||	Short-circuit OR
?:	Operator ternary if-then-else

PRESEDEN OPERATOR

Tertinggi			
()	[]	.	
++	--	~	!
*	/	%	
+	-		
>>	>>>	<<	
>	>=	<	<=
==	!=		
&			
^			
|			
&&			
||			
?:			
=	op=		
Terendah			

TIPE DATA

Java merupakan contoh bahasa yang *strongly typed language*.

Hal ini berarti bahwa setiap variabel harus memiliki tipe yang sudah dideklarasikan.

Terdapat 8 tipe primitif :

a) 6 diantaranya adalah tipe bilangan
   - 4 tipe integer
   - 2 tipe floating point

b) 1 tipe karakter char, digunakan untuk encode Unicode

c) 1 tipe boolean.

INTEGER

Tipe	Tempat yang Diperlukan	Jangkauan (inclusive)
int	4 byte	- 2.147.483.648 sampai 2.147.483.647 (hanya lebih dari 2 miliar)
short	2 byte	- 32.768 sampai 32.767
long	8 byte	- 9.223.372.036.854.775.808L sampai 9.223.372.036.854.775.807L
byte	1 byte	- 128 sampai 127

FLOATING POINT

Tipe	Tempat Yang Dibutuhkan	Jangkauan
float	4 byte	secara kasar 3,40282347E+38F (7 digit desimal signifikan)
double	8 byte	secara kasar ± 1,79769313486231570E+308 (15 digit desimal siignifikan)

CHAR

Tipe char menggunakan *tanda kutip tunggal* untuk menyatakan suatu *char*.

Tipe char juga menyatakan karakter dalam upaya encoding unicode, yang merupakan kode 2-byte.

Karakter unicode paling sering dinyatakan dalam istilah skema pengkodean hexadesimal yang dimulai dari \u0000 sampai \uFFFF.

Selain karakter bebas (escape ‘ \u ‘) yang menyatakan karakter unicode di Java terdapat juga

\b	backspace	\u0008
\t	tab	\u0009
\n	linefeed	\u000a
\r	carriage return	\u000d
\”	double quote	\u0022
\’	single quote	\u0027
\\	a backslash	\u005c

BOOLEAN

Tipe boolean memiliki nilai ***true*** *dan* ***false***.

Tipe ini digunakan untul logical testing dengan menggunakan operator relasional.

KONVERSI ANTAR NILAI NUMERIK

Operasi biner apapun pada variabel numerik dengan tipe yang berbeda dapat diterima dan diperlakukan dengan cara seperti dibawah ini :

1. Jika tipe operand adalah double, maka yang lain juga akan diperlakukan sebagai double pada lingkup operasi tersebut.
2. Jika operand adalah float, maka yang lain juga akan diperlakukan sebagai float.
3. Jika operand adalaha long, maka yang lain juga akan diperlakukan sebagai long.

Konversi yang diijinkan adalah sebagai berikut :

byte → **s**hort → **i**nt → **l**ong → **f**loat → **d**ouble

Dimana kita dapat memberikan nilai variabel suatu tipe disebelah kiri *ke tipe disebelah kanannya*.